In-Circuit Emulation for the Apple II Computer

You can convert your Apple into a host for testing a target system's hardware and software

by John D. Ferguson

Without doubt, the easiest method of debugging any microprocessor-based computer system is to access that system through its processor socket. Using in-circuit-emulation (ICE) techniques, the most powerful diagnostic aids currently available, the processor of the unit under test (the target) is removed, and a second microcomputer system (the host) is linked to the target through the target's processor socket.

Emulators are usually associated with expensive microprocessor development systems or equally expensive troubleshooting tools such as the Fluke 9010A (manufactured by the John Fluke Company of Everett, Washington). However, with one simple circuit you can turn your Apple II into a host computer for emulation purposes. This allows it to test hardware and evaluate software in a target system based on a 6502 microprocessor (or one compatible with the 6502).

The Apple II in-circuit emulator is carried on one Apple card. A 40-conductor ribbon cable, terminated in a dual-inline plug, connects the ICE to the target microcomputer (see figure 1). The ICE card gives the Apple limited emulation capability, allowing it to relocate any 2K-byte block of address space in the target system into the normally free memory area at locations C800 through CFFF hexadecimal in the Apple (see figure 2). The memory region observed in the target system is software selected by writing to an address-select latch. Because this selection is under program control, you can write routines in the host system to test the target system's entire memory map.

With one simple circuit you can turn the Apple II into a host computer for emulation purposes.

Test software can be written in either a high-level language, such as BASIC, or in machine code, and it can be directed at the main functional blocks within the target system: system buses, RAM (random-access read/write memory), ROM (read-only memory), and I/O (input/output) devices.

Routines written in BASIC tend to be inconveniently slow for even the simplest tests. The more effective approach is to write standard test modules in machine code and use a BASIC program to form an overall test strategy that sequences the tests and guides the user with recommendations if a fault is detected.

In this article I'll describe test modules for exercising the system buses and testing RAM and ROM. I'll conclude with a case study that illustrates how the Apple ICE can be used to test Rockwell International's AIM-65 single-board computer.

ICE Hardware

Figure 3 on page 422 shows a circuit diagram of the ICE card. Address lines A0 to A10 together with control lines R/$\overline{W}$, ϕ_0, and $\overline{RES}$ pass directly from the Apple to the target system via octal driver chips IC4 and IC5. However, address lines A11 to A15 in the target system are not obtained from their Apple equivalents but are instead generated by the block-select latch IC3. For selection of the five most significant lines in the target system, a control word is first written to this latch, which is clocked by the Apple I/O SELECT line. Hence, if the ICE was in slot 5, the following short program would set A11 to A15 in the target system to zero:

```
LDA #$00  \ sets A11 to A15 to zero
STA C500  \ activates the I/O SELECT
            line in slot 5.
```

After the block-select latch is con-

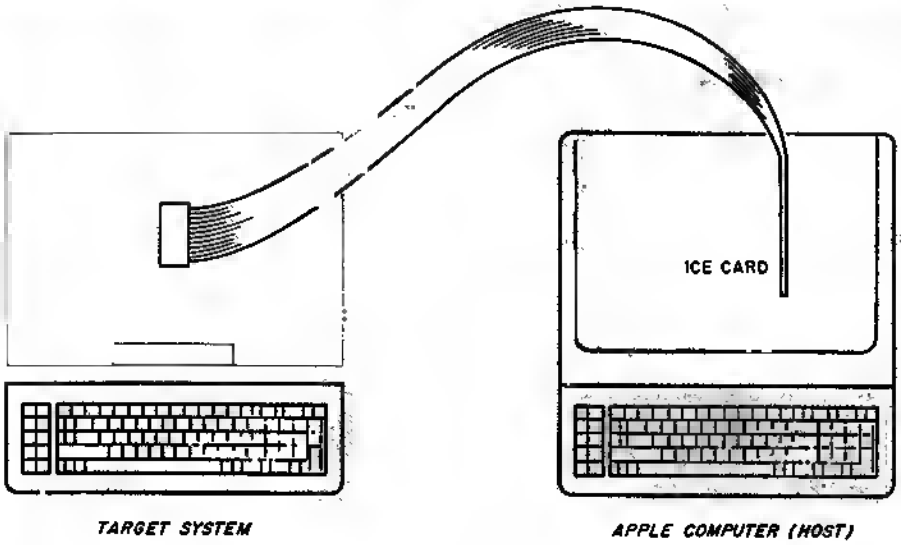


Figure 1: *The processor is removed from the target system, which is then connected to the host via a 40-conductor ribbon cable.*

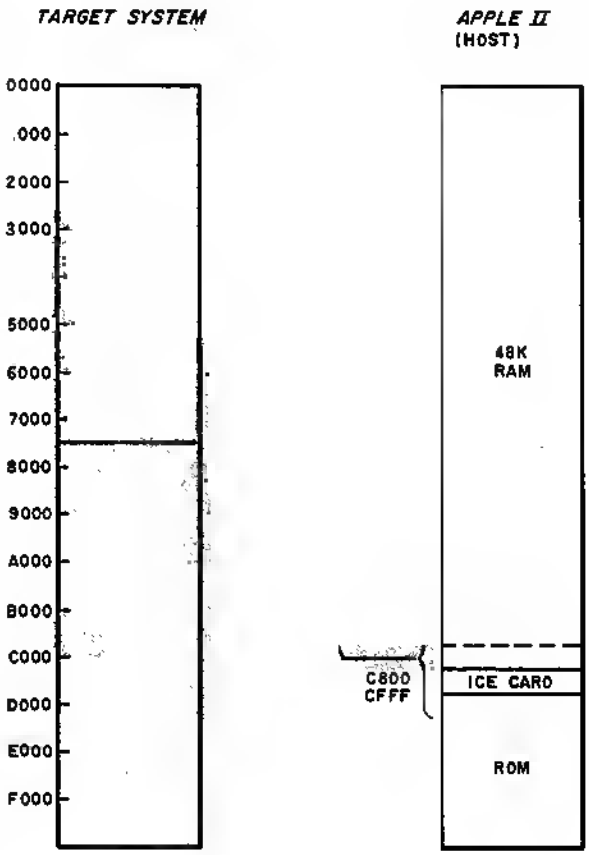


Figure 2: *Any 2K-byte block of memory space in the target system can be mapped into the normally empty memory slot from location C800 to CFFF hexadecimal in the Apple.*

figured, any read or write operations to memory locations between C800 and CFFF hexadecimal in the Apple activate the address decoder chip IC1 and enable the output of latch IC3, establishing corresponding addresses between 0000 and 07FF hexadecimal in the target system.

The address decoder (IC1) also enables the octal transceiver (IC2), allowing data to be either written to or received from the target system. The decoder IC1 might seem unnecessary because $\overline{\text{I/O STROBE}}$ is active low for addresses between C800 and CFFF hexadecimal. However, close examination of $\overline{\text{I/O STROBE}}$'s timing shows that its low state appears too late in the timing cycle to enable slow memory or I/O devices in the target system (see figure 4 on page 427).

Test Software

The software required for testing falls into two categories: (1) routines that exercise and test the various functional areas of the target microcomputer—its system buses, RAM, ROM, and I/O devices—and (2) the overall test program, which guides you through the test sequence, calling the functional tests and performing the tasks normally performed by a fault-finding tree (i.e., pinpointing the source of the fault and suggesting a remedy—for instance, "replace IC28"—or initiating a new test to gather more information).

The following section describes the functional tests, providing three routines written in 6502 assembly language. Each program operates on the memory window at locations C800 to CFFF hexadecimal between the Apple and the target system.

Address and Data-Bus Toggle Test

Before launching into complex tests of the system's ICs, test the integrity of the system buses. With a toggle test you can exercise the address and data-bus lines by alternately driving them high and low. Listing 1 on page 427 shows such a test program, which starts by selecting addresses in the binary pattern 10101 . . . in the target system. A dummy read is then made to address AAAA hexadeci-

Text continued on page 427

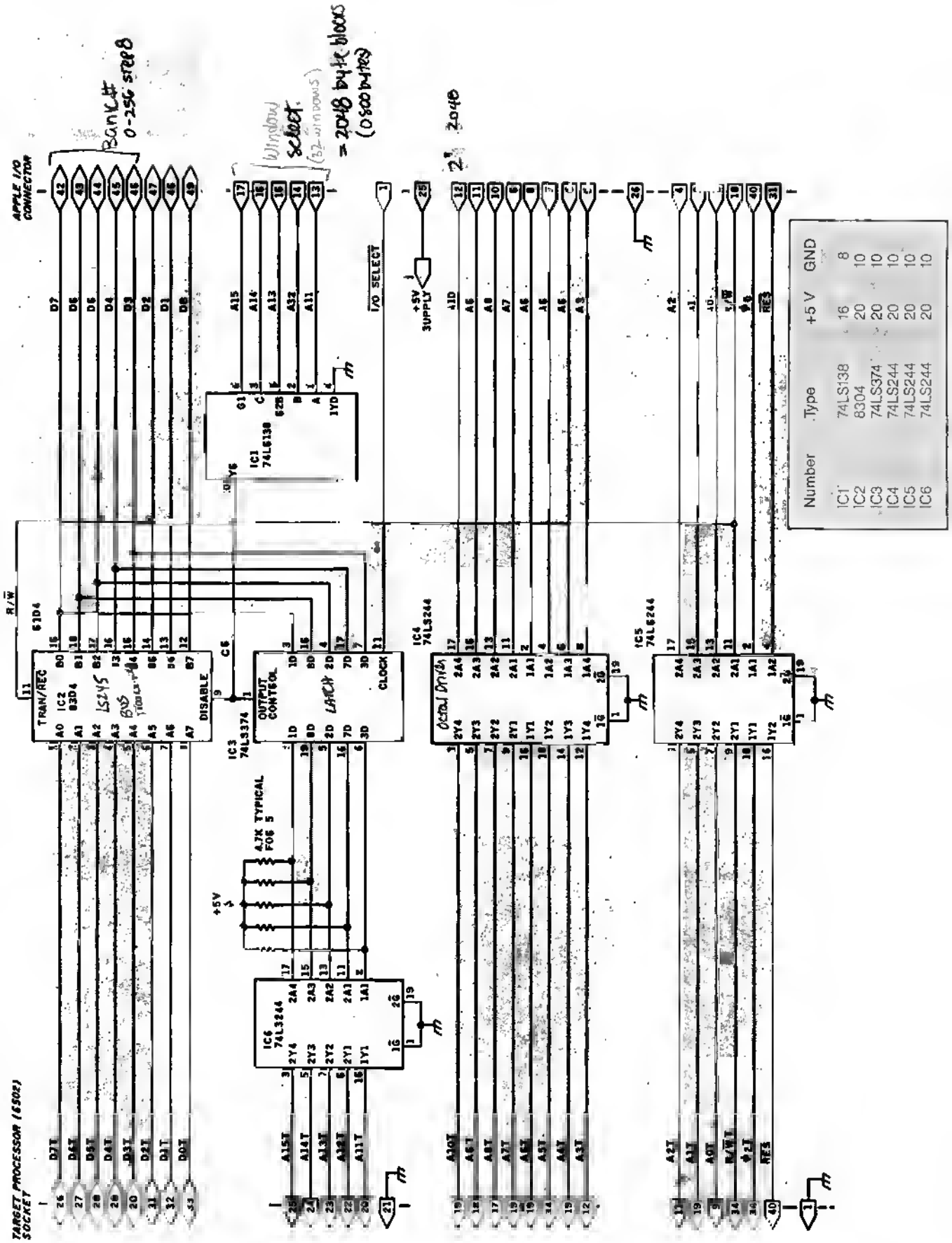


Figure 3: *A circuit diagram of the Apple ICE card.*

Text continued from page 420

mal, placing the high, low, high, low pattern on the target-system bus. The select latch is again accessed and addresses in the binary pattern 01010 . . . are selected, followed by a dummy read to location 5555 hexadecimal, thus complementing the previous address-bus pattern. This procedure is repeated 256 times before a similar test pattern is started on the target system's data bus.

Exercising the system buses with this pattern allows the operator to determine, using an oscilloscope or logic probe, whether each line in the target system is drivable (i.e., no lines are stuck high or low) and whether each line is continuous from its source (the processor socket) to its destination on each chip.

A more complex test could also check for shorts between lines by injecting a characteristic frequency or pattern onto each line. You could then use a frequency meter or oscilloscope to check for corruption between lines.

Toggling the system buses 256 times does not allow enough time for checking even one circuit node. The short routine below illustrates how the bus test (BTEST) is used in the test sequencing program:

```
180 PRINT "BUS TESTING—
    PROBE TARGET SYSTEM
    BUSES"
190 PRINT "(PRESS SPACE FOR
    NEXT TEST)"
200 CALL BTEST
210 IF PEEK (-16384) < =127
    THEN 200
```

By placing the test within a loop that also checks the keyboard, the test repeats until you press the space bar, signaling that further testing isn't needed.

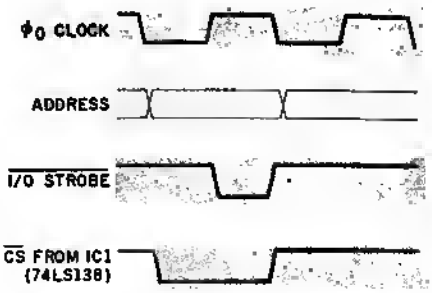


Figure 4: *Relative timing of the Apple slot signal I/O STROBE and the CS signal from IC1. Note that the I/O STROBE signal occurs too late for use in selecting the target address bus.*

Listing 1: *Toggling all address and data-bus lines.*

```
SOURCE FILE: APPTOG
0000:                  ********************************************************************************
0000:                  ADDRESS AND DATA BUS TEST
0000:                3 ; TOGGLE ADDRESS BUS AAAA-5555
0000:                4 ; 256 TIMES
0000:                5 ; TOGGLE DATA BUS AA-55
0000:                6 ; 256 TIMES
0000:                7 ;********************************************************************************
0000:                8 ;
C500:                9 SELECT        EQU     $C500       ;2K SELECT LATCH
——— NEXT OBJECT FILE NAME IS APPTOG.OBJ0
2100:               10               ORG     $2100
2100:A2 00          11               LDX     #00         ;SET COUNTER TO ZERO
2102:               12 ;EXERCISE ADDRESS BUS
2102:A9 AA          13 ABUS          LDA     #$AA        ;SELECT ADDRESSES 10101XXX
2104:8D 00 C5       14               STA     SELECT
2107:AD AA CA       15               LDA     $CAAA       ;E AAAA ON TARGET BUS
210A:A9 55          16               LDA     #$55        ;SELECT ADDRESSES 01010XXX
210C:8D 00 C5       17               STA     SELECT
210F:AD 55 CD       18               LDA     $CD55       ;E 5555 ON TARGET BUS
2112:CA             19               DEX
2113:D0 ED          20               BNE     ABUS        ;REPEAT 256 TIMES
2115:               21 ;EXERCISE DATA BUS
2115:A9 00          22 DBUS          LDA     #$00        ;SELECT ADDRESSES 00000XXX
2117:8D 00 C5       23               STA     SELECT
211A:A9 55          24               LDA     #$55        ;01010101 ON DATA BUS
211C:8D 00 C9       25               STA     $C900       ;0100 IN TARGET SYSTEM
211F:A9 AA          26               LDA     #$AA        ;10101010 ON DATA BUS
2121:8D 00 C9       27               STA     $C900       ;0100 IN TARGET SYSTEM
2124:CA             28               DEX
2125:D0 EE          29               BNE     DBUS        ;REPEAT 256 TIMES
2127:60             30               RTS                 ;TOGGLING COMPLETE

SUCCESSFUL ASSEMBLY: NO ERRORS

2102 ABUS           2115 DBUS        C500 SELECT         2102 ABUS
2115 DBUS           C500 SELECT
```

RAM Checkerboard Test

The basic strategy for testing RAM requires writing a test pattern into memory, reading it back, and checking that both the write and read operations were successful. Many different test patterns can be used; each is sensitive to particular failure modes of the memory. One popular pattern that provides in a reasonable amount of time a test of the read/write capability of every bit in the RAM is the checkboard test pattern (see figure 5).

Listing 2 shows this RAM test program. In it, a RAM location is selected and 55 hexadecimal (01010101 binary) is stored in the location and then read back and compared. If the comparison fails, the test terminates with the Apple displaying a RAM failure message. If the comparison passes, the location is then tested with the complementary pattern AA hexadecimal (10101010 binary). The test then moves on to the next location and continues until all locations within the window (C800 to CFFF hexadecimal) have been exercised and tested.

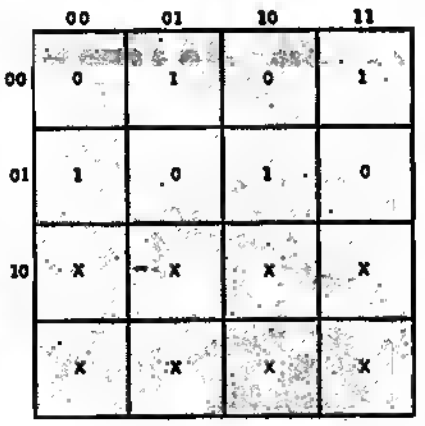


Figure 5: *Checkerboard-testing RAM. Alternate bits are set to 1 and 0 and checked. The pattern is then reversed and checked before moving off to the next row of cells.*

Before the RAM test in the main test program is called, the memory-select latch should be written to, moving the RAM to be exercised into the ICE test window. For example, the following program would test RAM from 0800 to 0FFF hexadecimal in the target system:

```
250 PRINT "RAM TESTING
    0800—0FFF"
260 POKE SELECT, 08:CALl
    RAMTEST
```

ROM Signatures

The usual method for testing ROMs involves forming a checksum byte based on a sum of all the data within the ROM. However, faults could be concealed by several errors that cancel each other out. A technique that is more sensitive and less likely to mask errors involves performing a cyclic redundancy check (CRC) on the ROM contents. It originated in data communications, but more recently it's been used in signature analysis, a relatively new troubleshooting tool pioneered by Hewlett-Packard. Like most jobs in computing, the cyclic redundancy check can be evaluated by either hardware or software. The hardware model proves the simplest to illustrate.

Figure 6 on page 435 shows a typical CRC evaluation circuit using a 16-bit linear shift register with feedback. Each bit of data is fed serially into the register. When the data stream ends, the final binary pattern remaining in the register forms the 4-digit cyclic redundancy check. The feedback paths effectively form a sum to the base 2 between the data fed back and the new data entering and ensure that every bit entering the register contributes toward the final CRC or signature.

An equivalent software routine is presented in listing 3. In this scheme, each byte from the ROM under test is fed serially (bit 0 to bit 7) to the subroutine FEEDBACK, which performs a sum to the base 2 of bits 15, 11, 8, and 6 within the register and the incoming bit. When 16,384 (2K × 8) bits of data have entered the feedback algorithm, the pattern remaining in locations SIGH and SIGL forms the final signature.

To enable checks to be made on ROMs containing more than 2K

Text continued on page 435

Listing 2: *A program to checkerboard-test RAM.*

```
SOURCE FILE: APPRAM
0000:                    1 ;*****************************************************************
0000:                    2 ;PROGRAM TO CHECKERBOARD TEST RAM
0000:                    3 ; (C800-CFFF)
0000:                    4 ;*****************************************************************
0000:                    5 ;
FDED:                    6 COUT      EQU   $FDED     ;CHARACTER TO SCREEN
FD8E:                    7 CROUT     EQU   $FD8E     ;C-RETURN TO SCREEN
FDE3:                    8 PRHEX     EQU   $FDE3     ;OUTPUT HEX DIGIT
0008:                    9 POINT     EQU   08        ;POINTER
0000:                   10 ;
----- NEXT OBJECT FILE NAME IS APPRAM.OBJ0
2090:                   11           ORG   $2090
2090:                   12 ;
2090:A9 00              13           LDA   #00       ;POINT TO C800
2092:85 08              14           STA   POINT
2094:A8                 15           TAY
2095:A9 C8              16           LDA   #$C8
2097:85 09              17           STA   POINT+1
2099:A9 55              18 START     LDA   #$55      ;START TEST WITH 55
209B:91 08              19           STA   (POINT),Y ;STORE
209D:D1 08              20           CMP   (POINT),Y ;READ BACK AND COMPARE
209F:F0 03              21           BEQ   OK
20A1:4C BC 20           22           JMP   ERROR     ;DISPLAY ERROR MESSAGE AND END
20A4:A9 AA              23 OK        LDA   #$AA      ;NOW TRY AA
20A6:91 08              24           STA   (POINT),Y ;STORE
20A8:D1 08              25           CMP   (POINT),Y ;READ BACK AND COMPARE
20AA:F0 03              26           BEQ   OK1
20AC:4C BC 20           27           JMP   ERROR     ;DISPLAY ERROR MESSAGE AND END
20AF:E6 08              28 OK1       INC   POINT     ;NEXT LOCATION
20B1:D0 E6              29           BNE   START
20B3:E6 09              30           INC   POINT+1
20B5:A5 09              31           LDA   POINT+1
20B7:C9 D0              32           CMP   #$D0
20B9:D0 DE              33           BNE   START     ;END OF BLOCK?(CFFF)
20BB:60                 34           RTS             ;TEST COMPLETE
20BC:                   35 ;
20BC:                   36 ;ERROR DISPLAY ROUTINE
20BC:                   37 ;
20BC:A2 00              38 ERROR     LDX   #00       ;POINTER FOR MESSAGE
20BE:BD D5 20           39 NEXT1     LDA   MESS,X
20C1:20 ED FD           40           JSR   COUT      ;MESSAGE TO SCREEN
20C4:E8                 41           INX             ;NEXT CHARACTER
20C5:E0 0F              42           CPX   #$0F      ;MESSAGE COMPLETE?
20C7:D0 F5              43           BNE   NEXT1
20C9:A5 09              44           LDA   09        ;FAIL ADDRESS TO SCREEN
20CB:38                 45           SEC
20CC:E9 C8              46           SBC   #$C8
20CE:20 E3 FD           47           JSR   PRHEX
20D1:20 8E FD           48           JSR   CROUT     ;C-RETURN TO SCREEN
20D4:60                 49           RTS             ;  AND FINISHED
20D5:                   50 ;ERROR MESSAGE
20D5:A0 C5 D2           51 MESS      ASC   "         ERROR ON PAGE "
20D8:D2 CF D2
20DB:A0 CF CE
20DE:A0 D0 C1
20E1:C7 C5 A0

*** SUCCESSFUL ASSEMBLY: NO ERRORS
```

Text continued from page 428

bytes, three routines are used. The NSIG (new signature) routine resets the shift register pair SIGH, SIGL to zero and forms a signature on 2K bytes of ROM. In CSIG (continue signature), the shift register is *not* reset to zero at the start, thus allowing a continuation of a signature for ROMs greater than 2K bytes. The DISPLAY routine shows the contents of the shift register pair SIGH, SIGL in hexadecimal form.

The following listing shows how all three routines can be used to evaluate the signature of a 4K-byte ROM located at B000 to BFFF hexadecimal in the target system:

```
320 REM B0 HEX IS 176
    DECIMAL
330 POKE SELECT,176
340 CALL NSIG:REM FIRST 2K -
    BYTES
350 REM B8 HEX IS 184
    DECIMAL
360 POKE SELECT,184
370 CALL CSIG:REM CONTINUE
    WITH NEXT 2 BYTES
380 CALL DISPLAY:REM
    DISPLAY FINAL SIGNATURE
```

Implementing a Test Program

The Apple ICE described here can be used with a wide range of 6500 microcomputers designed to run at 1 MHz if all the onboard circuitry is controlled by the processor's ϕ_2 clock. The AIM-65, for example, provides an ideal target system, containing as much as 4K bytes of RAM, 20K bytes of ROM, and a wide range of I/O devices—two 6522 VIAs (versatile interface adapters), a 6520 PIA (peripheral interface adapter), and a 6532 RIOT (RAM input/output timer). Figure 7 provides an overview and a memory map of the AIM-65, and a test sequence is shown in listing 4 on page 443. The program begins by testing the system buses, followed by a RAM test on the 4K bytes of RAM and a ROM test that forms signatures for each of the five system ROMs. The test sequence concludes with a check on the user 6522 VIA. For this test, the ports are linked together

Text continued on page 444

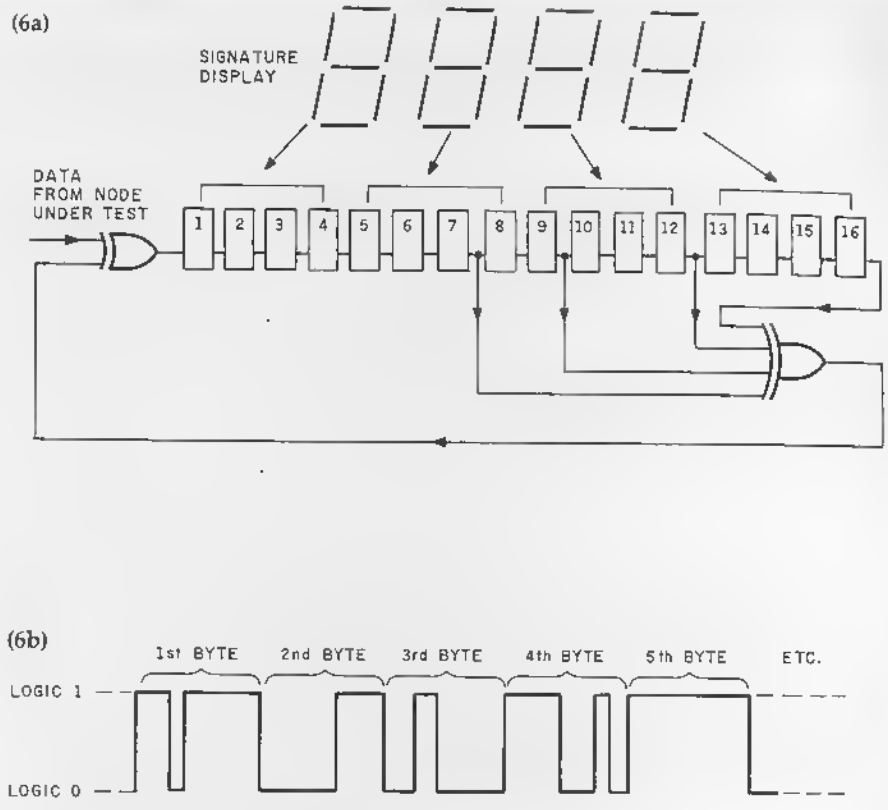
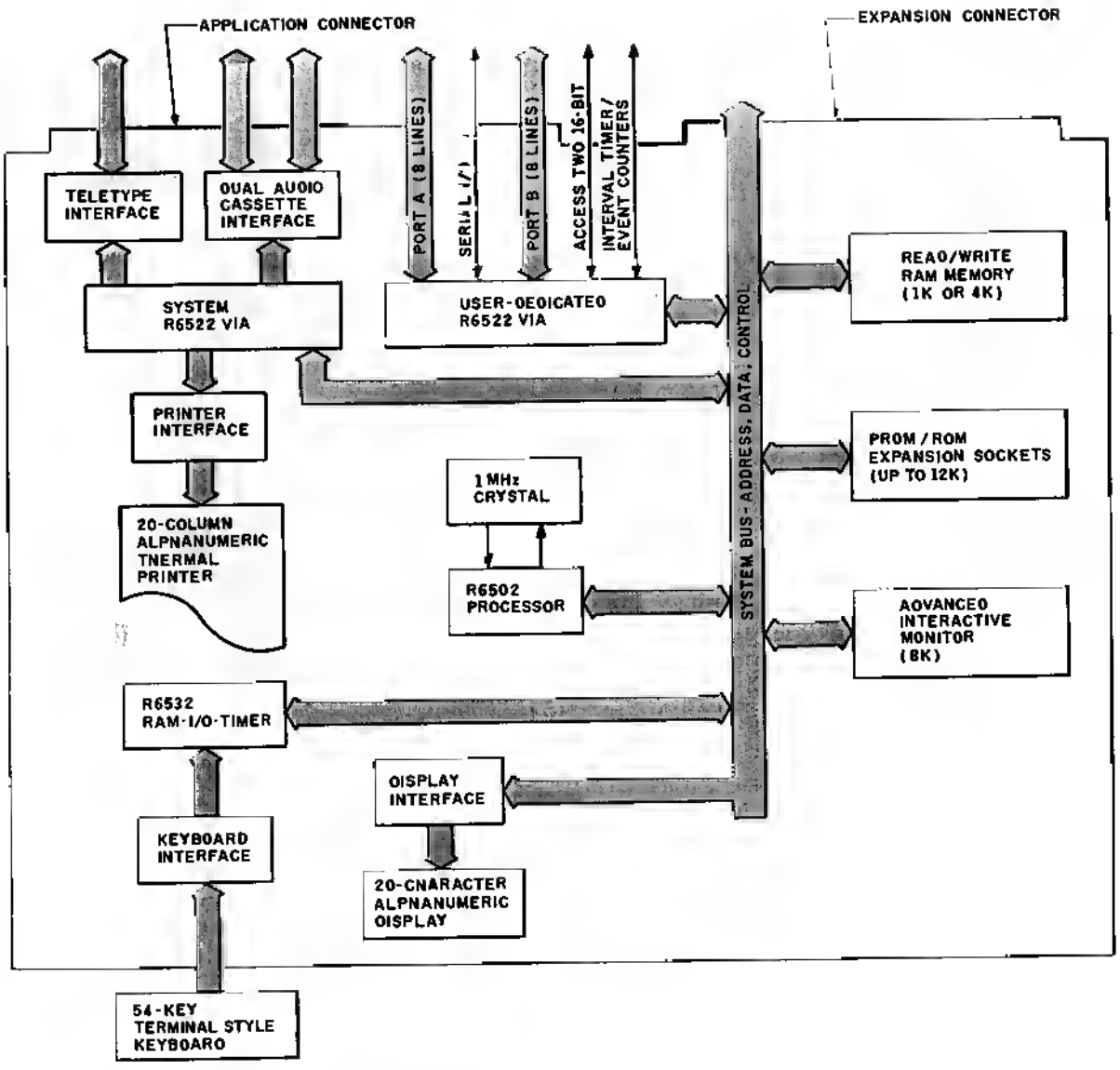


Figure 6: *A feedback shift register can be used to form a cyclic redundancy check of ROM (6a). Data from ROM is fed in bit-serial (bit 0 to bit 7) byte-serial form into the shift register (6b).*



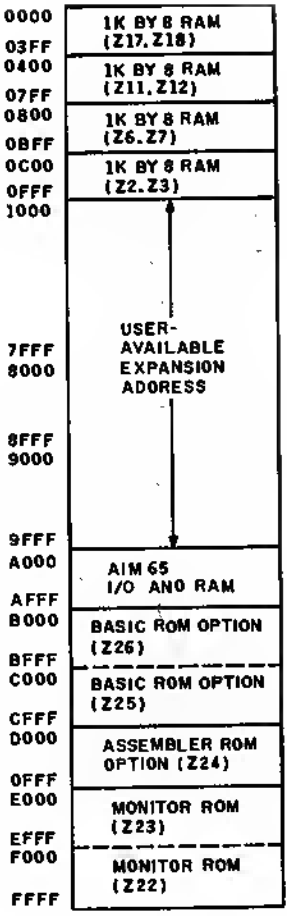


Figure 7: *The layout and memory map of the AIM-65.*

Listing 3: *A program to evaluate ROM signatures.*

```
SOURCE FILE: APPSIG
0000:                    1 ;*****************************************************************
0000:                    2 ; PROGRAM TO EVALUATE SIGNATURE
0000:                    3 ;OF 2KBYTE BLOCK (C800-CFFF)
0000:                    4 ;EACH BYTE IS SERIALIZED BIT0-BIT7
0000:                    5 ;*****************************************************************
0000:                    6 ;
0000:                    7 ;
0000:                    8 ;
———— NEXT OBJECT FILE NAME IS APPSIG.OBJ0
2000:                    9                ORG     $2000
1900:                   10 COUNT          EQU     $1900       ;STORE FOR SUM
1901:                   11 SIGL           EQU     $1901       ;CURRENT SIGNATURE LOW BYTE
1902:                   12 SIGH           EQU     $1902       ;CURRENT SIGNATURE HIGH BYTE
0008:                   13 POINT          EQU     $0008       ;BYTE COUNTER
1903:                   14 TEMP           EQU     SIGH+1      ;TEMPORARY STORE
FDDA:                   15 PRBYTE         EQU     $FDDA       ;PRINT A HEX BYTE
FD8E:                   16 CROUT          EQU     $FD8E       ;GENERATE C-RETURN
2000:                   17 ;
2000:A9 00              18 START          LDA     #00         ;ZERO SHIFT REGISTER
2002:8D 01 19           19                STA     SIGL
2005:8D 02 19           20                STA     SIGH
2008:A9 00              21 WSTART         LDA     #00         ;WARM START
200A:85 08              22                STA     POINT
200C:A8                 23                TAY
200D:A9 C8              24                LDA     #$C8        ;START OF BLOCK C800
200F:85 09              25                STA     POINT+1
2011:BI 08              26 NBYTE          LDA     (POINT),Y   ;GET BYTE
2013:8D 03 19           27                STA     TEMP
2016:A2 08              28                LDX     #08         FOR 8 BITS
2018:AD 03 19           29 NBIT           LDA     TEMP
201B:29 01              30                AND     #01         BIT0 INTO COUNT
2010:8D 00 19           31                STA     COUNT
2020:20 36 20           32                JSR     FEEDBACK    ;APPLY FEEDBACK
2023:6E 03 19           33                ROR     TEMP        ;READY FOR NEXT BIT
2026:CA                 34                DEX
2027:D0 EF              35                BNE     NBIT        ;BACK FOR NEXT BIT
2029:E6 08              36                INC     POINT       ;NEXT BYTE
202B:D0 E4              37                BNE     NBYTE
2020:E6 09              38                INC     POINT+1
202F:A5 09              39                LDA     POINT+1
2031:C9 D0              40                CMP     #$D0
2033:D0 DC              41                BNE     NBYTE       END OF BLOCK? CFFF
2035:60                 42                RTS
2036:                   43 ;
2036:                   44 ;
2036:                   45 ;FEEDBACK ALGORITHM—SUMS BITS
2036:                   46 ;15,11,8 AND 6 WITH INCOMING BIT
2036:                   47 ;ON ENTRY 'COUNT' CONTAINS INPUT BIT
2036:                   48 ;
2036:AD 02 19           49 FEEDBACK       LDA     SIGH        ;TOP HALF OF SIG
2039:10 03              50                BPL     NEX1        ;TEST BIT15
203B:EE 00 19           51                INC     COUNT
203E:6A                 52 NEX1           ROR     A
203F:90 03              53                BCC     NEX2        ;TEST BIT 8
2041:EE 00 19           54                INC     COUNT
2044:6A                 55 NEX2           ROR     A
2045:6A                 56                ROR     A
2046:6A                 57                ROR     A
2047:90 03              58                BCC     NEX3        ;TEST BIT
2049:EE 00 19           59                INC     COUNT
204C:AD 01 19           60 NEX3           LDA     SIGL        ;BOTTOM HALF OF SIG
204F:2A                 61                ROL     A
2050:2A                 62                ROL     A
2051:90 03              63                BCC     NEX4        ;TEST BIT 6
2053:EE 00 19           64                INC     COUNT
2056:6E 00 19           65 NEX4           ROR     COUNT       ;SUM INTO CARRY
2059:2E 01 19           66                ROL     SIGL        ;CARRY INTO BIT0 LBYTE
```

Listing 3 continued on page 443

Listing 3 continued:

```
205C:2E 02 19    67             ROL     SIGH    ;CARRY INTO BIT0 HBYTE
205F:60          68             RTS
2060:AD 02 19    69  DISPLAY     LDA     SIGH    ;MSB TO DISPLAY
2063:20 DA FD    70             JSR     PRBYTE  ;ONTO APPLE DISPLAY
2066:AD 01 19    71             LDA     SIGL    ;LSB TO DISPLAY
2069:20 DA FD    72             JSR     PRBYTE  ;ONTO APPLE DISPLAY
206C:20 8E FD    73             JSR     CROUT   ;C-RETURN
206F:60          74             RTS

* SUCCESSFUL ASSEMBLY; NO ERRORS
```

Listing 4: *Applesoft BASIC program sequencing tests.*

listing 4

```
50          REM AIM65 TEST ROUTINE
60          HOME
70          REM DEFINE SYSTEM ADDRESSES
80        SELECT = - 15100  50432=C500 -slot5 select
90        DISPLAY = 8288
100         NSIG = 8192
110         CSIG = 8200
120         BTEST = 8448
130         RAMTEST = 8336
140         PRINT "  "
150         PRINT "LOADING MACHINE CODE TESTS"
160         PRINT "BLOAD APPTESTS"
170         PRINT "  "
180         PRINT "BUS TESTING-PROBE TARGET SYSTEM BUSES"
190         PRINT "(PRESS SPACE FOR NEXT TEST)"
200         CALL BTEST
210         IF PEEK (- 16384) < = 127 THEN 200
220         PRINT "  "
230         PRINT "RAM TESTING 0000-07FF"          0-256 step 8
240         POKE SELECT,0: CALL RAMTEST
250         PRINT "RAM TESTING 0800-0FFF"
260         POKE SELECT,08: CALL RAMTEST
270         PRINT "  ": PRINT " RAM TESTS COMPLETE "
280         PRINT "  "
290         PRINT "ROM SIGNATURES BLOCKS B,C,D,E,F "
300         PRINT "  "
310         FOR N = 176 TO 240 STEP 16
320         POKE SELECT,N: CALL NSIG
330         POKE SELECT,(N + 8): CALL CSIG
340         CALL DISPLAY
350         NEXT N
360         PRINT "  ": PRINT " ROM SIGNATURES COMPLETE"
370         PRINT "  "
380         PRINT " VIA TEST"
390         POKE SELECT,160: REM SELECT BLOCK AXXX
400       APRT = 51201:BPRT = 51200
405       ADIR = 51203:BDIR = 51202
410         POKE ADIR,0: POKE BDIR,255
415         REM A INPUT - B OUTPUT
420         FOR N = 0 TO 255
430         POKE BPRT,N
440         IF PEEK(APRT) < > N THEN PRINT "VIA ERROR"
450         NEXT N
460         POKE BDIR,0: POKE ADIR,255
465         REM BINPUT - A OUTPUT
470         FOR N = 0 TO 255
480         POKE APRT,N
490         IF PEEK (BPRT) < > N THEN PRINT "VIA ERROR"
500         NEXT N
510         PRINT "  ": PRINT " TEST COMPLETE"
520         END
```

Listing 4 continued on page 444

Listing 4 continued:

```
]RUN

LOADING MACHINE CODE TESTS

BUS TESTING-PROBE TARGET SYSTEM BUSES   (PRESS SPACE FOR NEXT TEST)

RAM TESTING 0000-07FF
RAM TESTING 0800-0FFF

  RAM TESTS COMPLETE

ROM SIGNATURES BLOCKS B,C,D,E,F

B89C
A181
F727
B072
8A9E

  ROM SIGNATURES COMPLETE

  VIA TEST

  TEST COMPLETE
```

Text continued from page 435:

with a hard-wired fixture connecting PA0 to PB0, PA1 to PB1, and so on. The routine starts by configuring port A as an input and port B as an output. A test pattern is then written out port B and read and checked at port A. The role of the ports is then reversed, and the test is repeated.

This example illustrates some techniques that can be used with the Apple ICE. A more detailed program for the AIM-65 would test the remaining I/O devices, such as the display, printer and keyboard, and more thoroughly guide the user. However, the ideas presented here illustrate the principles behind the techniques and mirror those found in commercial instruments. The Apple ICE is therefore not only a practical fault-finding tool but also an ideal, low-cost, educational aid.■

John D. Ferguson is a lecturer with the Microelectronics Educational Development Centre at Paisley College, High St., Paisley PA1 2BE, Scotland.